Desarrollo de un selector de imágenes según su nitidez

Joan Treveset Vázquez

Resumen — Este proyecto consiste en el desarrollo de un selector de imágenes con el objetivo de automatizar, simplificar y acelerar la selección de imágenes según su nitidez. Concretamente estamos estudiando el caso de varias imágenes de la misma escena y con el mismo encuadre. Se implementa un aplicación escrita en C++ con la ayuda de las librerías QT y libTIFF para ordenadores de escritorio. En cuanto a la evaluación de la nitidez se utiliza una implementación propia del gradiente de la imagen. Se han llevado a cabo pruebas y mientras la métrica escogida es bastante acertada también es muy sensible a la región de la imagen que se evalúe. Por este motivo se ha investigado en alguna manera de localizar una región de interés. Para intentar solucionar este problema han probado dos métodos, ambos basados en la extracción y correlación de características pero ambos han resultado ser insuficientemente precisos para esta aplicación. El rendimiento de la aplicación en un caso real dependerá de cuan descuadradas se encuentren las imágenes entre sí.

Palabras clave — Evaluación de nitidez, Gradiente, Interfaces de usuario gráficas, Localización de RoI

Abstract — This project consists of the development of an application that will automate, simplify and speed-up the image-selection process by measuring and comparing image sharpness. If given a set of digital photographic images of the same scene and with the same framing are taken, an application written in C++ is implemented using QT and libTIFF libraries targeting desktop computers. In order to measure and compare the sharpness of the image set, we use a custom implementation of the image gradient. In preliminary experiments it was noted that while the metric performs nicely for this application, it is very sensitive to the specific region of the image being evaluated. Because of this we have tried To address this issue, two different methods of locating the Region of Interest were tested, both based on feature extraction and matching. However, both methods yielded poor results for our purposes. The performance of the application in a real case scenario will depend on how different the framing of one image is from the subsequent one.

Index Terms — Gradient, Graphical user interfaces, Region of Interest location, Sharpness evaluation

1 Introducción

MUCHAS veces al hacer fotos nos encontramos con que salen desenfocadas o movidas, ya sea por tomarlas a mano alzada o falta de iluminación. Ante esta situación los fotógrafos tienden a hacer varias instantáneas de la misma escena con la esperanza de que alguna foto sea aceptable en cuanto a nitidez. Posteriormente el trabajo del fotógrafo es seleccionar la mejor foto de entre todas las que ha sacado. Dicho trabajo puede ser laborioso y tedioso, sobre todo si trabajamos con un gran número de imágenes o de mucha resolución, puesto que será necesario hacer *zoom* para apreciar los detalles.

Con esta motivación como base se propone una aplicación para ordenadores de escritorio capaz de hacer esta selección. Con el aplicativo resultante de este proyecto el fotógrafo podrá automatizar el proceso de selección de imágenes ahorrándole tiempo en dicha tarea.
El concepto aquí presentado puede tener también aplicación en la fotografía móvil. Debido al reducido tamaño de un teléfono móvil actual las fotos en los mismos suelen necesitar mayor tiempo de exposición y por lo tanto son más propensas a resultar movidas. Este efecto se ve amplificado gracias al reducido peso de los terminales. Al tener porca inercia es muy fácil moverlo involuntariamente. Por lo tanto de la misma manera que un fotógrafo profesional toma diversas instantáneas para después seleccionar la mejor el usuario de un terminal podría hacer lo mismo, o que una aplicación lo hiciese por él.

2 Organizacion del documento

Este artículo final se estructura en diversas partes listadas a continuación.

1. Objetivos y requisitos. Se establecen los objetivos del proyecto, así como el análisis de su nivel de completitud. Además se especifica el alcance del

- *E-mail de contacto: j.v.treveset@gmail.com*
- *Mención realizada: Enginyeria de Computadors*
- *Trabajo tutorizado por: Francisco Javier Sánchez Pujadas (Ciencias de la Computación)*
- *Curso 2013 - 2014*

proyecto.
2. Estado del arte. Un breve repaso de las técnicas existentes para la evaluación de la nitidez en las imágnes.
3. Metodología.
4. Resultados. Se exponen los resultados del proyecto. Este apartado es dividido en los diferentes módulos de los cuales se constituye el programa. Se realiza también un análisis de los resultados.
5. Conclusiones.
6. Agradecimientos.
7. Referencias.

3 OBJETIVOS Y REQUISITOS

A continuación se listan y evalúan los objetivos marcados al inicio del proyecto.

Proporcionar una herramienta para la clasificación de imágenes según su nitidez.
Como veremos más adelante se ha podido completar el desarrollo de una aplicación capaz de evaluar la nitidez de las imágenes. Los detalles del funcionamiento se explican en la sección de resultados.

Disminuir el tiempo de selección de imágenes, que actualmente es un proceso manual.
Este es un punto muy importante a tener en cuenta. La aplicación tiene que reducir el tiempo empleado por el usuario para encontrar la mejor imagen. Al tratarse de una aplicación con potenciales usuarios profesionales se han usado para las pruebas imágenes de gran calidad. Dichas imágenes presentan una resolución de 36 millones de píxeles y una profundidad de color de 16 bits por muestra, generando así un tamaño cercano a los 200 MB. Por este motivo las operaciones que más tiempo requieren son las de lectura de imágenes. Pese a ello, se considera que el objetivo ha sido cumplido satisfactoriamente.

Ofrecer en el aplicativo una interfaz de usuario agradable e intuitiva (*user friendly*).
Se ha intenado, en la medida de lo posible, ofrecer una experiencia de usuario eficaz. Dado que no se ha tenido ocasión de realizar pruebas de usuario la evaluación de este objetivo es subjetiva.

Por otro lado, al comienzo del proyecto se plantearon además diversos requisitos funcionales y no funcionales.

3.1 Requisitos funcionales

La aplicación estará enfocada a ordenadores personales.

Se implementará una interfaz gráfica de usuario que permita seleccionar la zona de la imagen en la cual se evaluará la nitidez.
Se ha desarrollado la aplicación con la ayuda de la librería gráfica QT en su versión de escritorio. Se ha elegido QT ya que ofrece un *framework* muy completo tanto para aspectos gráficos y de interfaz como en otras áreas tales como estructuras de datos o gestión de hilos de ejecución. QT es desarrollada por Nokia, es multiplataforma y de código libre. Actualmente la última versión estable es la 5.3 pero para este proyecto se ha utilizado la 5.1, ya que en el momento de su comienzo dicha versión era la más reciente.

La aplicación debe poder leer los formatos de imagen JPEG y TIFF de 8 y 16 bits por muestra.
Este requisito ha sido cubierto gracias a otra librería utilizada en la aplicación: libTIFF. El motivo de su uso es la incapacidad de QT para leer imágenes con una profundidad de color de 16 bits. Puesto que el *target* de la aplicación es el sector profesional creímos importante la inclusión de esta característica en la aplicación. Finalmente la aplicación es capaz de leer todos los formatos de imágenes que QT puede con la adición del formato tiff con 16 bits por muestra.

En cuanto a la valoración de la calidad de una imagen se evaluará la nitidez (enfoque) y trepidación (movimiento) de la misma.
Para la evaluación de la nitidez se utiliza el gradiente de la imagen. Hemos encontrado que la métrica utilizada teóricamente puede servir para evaluar también la trepidación en algunos casos, o bien dar falas lecturas en otros. Con las imágenes no sintéticas utilizadas para las pruebas los resultados han sido muy satisfactorios en este aspecto.

3.2 Requisitos no funcionales

La aplicación ha de ser portable al menos a dos sistemas operativos (Windows, Linux).
Aunque el desarrollo se ha llevado a cabo completamente en una plataforma Windows, se ha procurado no utilizar ninguna característica exclusiva de este sistema operativo, por lo que el código de la aplicación debería poder ser compilado para otras plataformas con mínimas modificaciones.

Estabilidad: el programa tendrá que hacer una mínima gestión de errores para garantizar que no falla bajo un uso normal.
No es posible garantizar una estabilidad absoluta del aplicativo, pero se ha procurado, en la medida de lo posible, hacer las comprobaciones necesarias y un mínimo *testing* de cada nódulo.
Para asegurar el cumplimiento de este requisito se han llevado a cabo diversas pruebas. Dichas pruebas consisten en comportamientos de usuario inesperados o ilógicos.

- Cargar imágenes y acto seguido volver a cargar otra vez.
- Seleccionar una miniatura que aun no esté cargada.
- Procesar el mismo conjunto de imágenes más de una vez sin recargarlo.
- Pulsar los botones de zoom antes de cargar una imagen en el área de previsualización.

- Seleccionar "Importar imagenes" y cerrar el "diálogo" sin seleccionarlas.
- Interacción mientras se están procesando imágenes. Se impide la interacción deshabilitando los botones de la interfaz excepto el Cancelar.
- Intentar procesar una sola imagen, en este caso no ocurre nada.

En todos estos casos la aplicación no falla y es posible continuar la ejecución.
Es sabido que "el testing nunca termina" en el mundo del software y por este motivo es probable que un usuario descubra un patrón de actuación con el programa que ocasione un mal funcionamiento del mismo.

El tiempo de ejecución ha de ser aceptable.

Requisito relacionado directamente con un objetivo del proyecto. Ha sido cumplido con creces. En caso de que la ejecución del programa resultase ser muy lenta sería por una baja relación entre la velocidad de lectura del dispositivo de almacenamiento y el tamaño de los ficheros. Puesto que podemos considerar que la velocidad de lectura es independiente de la aplicación, este tiempo también sería requerido por una simple aplicación de visualización de imágenes.

La interfaz de usuario tiene que ser intuitiva.

Aspecto más subjetivo del proyecto. Ante la incapacidad de efectuar *user testing* no es posible asegurar el cumplimiento de este objetivo.

Código abierto.

Tanto el lenguaje (C++) como las dos librerías usadas, QT y libTIFF, son de código abierto, por lo tanto este proyecto así lo es.

4. ESTADO DEL ARTE

Actualmente encontramos varias técnicas para evaluar la nitidez o el desenfoque de una imagen, mientras que cuantificar su movimiento o trepidación es una tarea más complicada.
La métrica más simple que podemos usar para cuantificar la nitidez es el gradiente de la imagen. Esto se basa en que en una imagen desenfocada las diferencias entre el valor de un pixel y sus vecinos son menores. Por lo tanto podemos asumir que cuanto más gradiente (mayor valor) tenga una región más enfocada estará.

$$M_{Gradiente} = \int\int \left|\frac{\partial g_i(x,y)}{\partial x}\right| + \left|\frac{\partial g_i(x,y)}{\partial y}\right| \mathrm{d}x\mathrm{d}y$$

Existen también otras métricas, aunque todas ellas se basan en la idea de enfatizar las altas frecuencias, es decir, los cambios bruscos de intensidad.
Una de las métricas más usadas y citadas es la propuesta por Subbarao et al. en 1993, que se basa en medir la varianza de los niveles de gris de la imagen. La diferencia respecto a la métrica del gradiente comentada antes radica en que cada píxel se compara con el color medio de los pixeles en una región.

$$M_{Subbarao} = \int\int (g_i(x,y) - m_i)^2 \mathrm{d}x\mathrm{d}y$$

La métrica utilizada ha sido una muy parecida a la primera citada. La principal diferencia radica en elevar al cuadrado el valor del gradiente, de esta manera tendremos más diferencia en el valor de la métrica para imágenes con diferente nivel de nitidez.

5. Metodología

La metodología seguida en el desarrollo del proyecto ha sido un desarrollo en espiral. Inicialmente se planteó dividir el programa en tres grandes módulos; interfaz, carga de imágenes y procesado, pero en el transcurso del proyecto surgió la necesidad de otro módulo, la gestión de hilos de proceso.
La primera versión del aplicativo fue escrita en MATLAB. Esto se hizo así ya que se trata de un lenguaje de muy alto nivel y que permite "maquetar" los conceptos. Dicha versión inicial resultó rudimentaria y el único propósito era confirmar la viabilidad del proyecto. A continuación se expone el pseudocódigo de dicha versión:

```
inicio
    lista <- LeerFicheros()
    puntos <- PedirCoordenadas()
    listaSubImagenes <- []
    listaGradientes <- []

    para imagen en lista
        listaSubImagenes.
        agregar(ExtraerSubImagen(puntos, imagen))
    finpara

    para subImagen en listaSubImagenes
        listaGradientes.
        agregar(gradiente(subImagen))
    finpara

    para gradiente en listaGradientes
        EscribirPorPantalla(gradiente)
    finpara

fin
```

Puesto que se trata de un desarrollo en espiral, podemos considerar esta la primera iteración. Con esta versión inicial funcional se procedió a la migración a C++.

El entorno de trabajo fue Microsof Visual Studio 2012 configurado con las librerías QT y libTIFF.

A la hora de hacer este paso a C++ se separó el programa en los tres módulos anteriormente comentados. Se creó una versión muy básica de cada módulo, con una funcionalidad mínima. El primero en desarrollar fue el de carga de imágenes con la ayuda de libTIFF, seguido del procesado del gradiente y por último una interfaz muy sencilla capaz únicamente de mostrar una imagen.

A partir de aquí se han ido desarrollando mejoras para cada módulo y agregando las características que se detallan en el apartado siguiente. En el transcurso del proyecto se originó la necesidad del desarrollo de un módulo no previsto inicialmente; la gestión de hilos de ejecución. Dicha necesidad se originó al apreciar la lentitud de carga de las imágenes. Ésta gestión de hilos se usa principalmente para cargar imágenes en segundo plano.

6. RESULTADOS

El resultado del proyecto es una aplicación capaz de informar al usuario cuál es la mejor imagen en términos de nitidez de un conjunto de imágenes de la misma escena.
La aplicación ha sido desarrollada completamente en C++ procurando escribir un código estructurado.

6.1 Funcionamiento de la aplicación e interfaz

En primer lugar, al abrir la aplicación se podrá observar la interfaz de la misma, tal y como se aprecia en la Fig. 1.

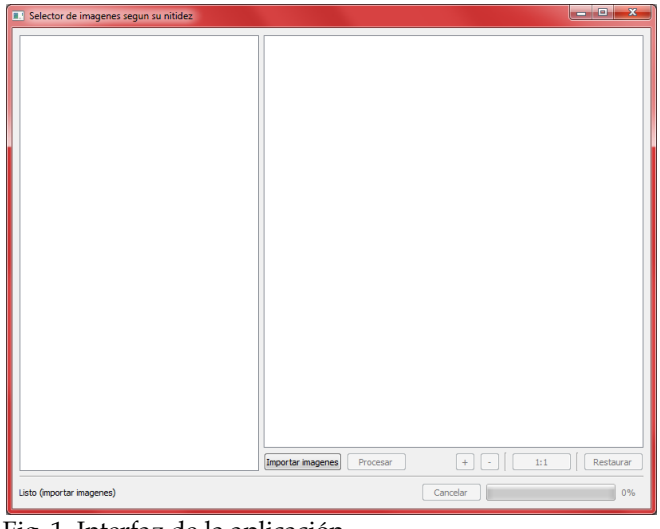

Fig. 1. Interfaz de la aplicación.

En esta captura podemos ver el diseño general de la aplicación.
La mayor cantidad de espacio se reserva para la visualización de la imagen seleccionada (una vez cargadas). En la parte izquierda encontramos otro espacio que se utiliza, tal y como puede apreciarse en la Fig. 2 para mostrar el listado de imágenes importadas.
Debajo del área principal tenemos la sección de controles. Éstos controles permiten al usuario llevar a cabo operaciones como Importar las imágenes o comenzar el procesado. Además se incluyen controles de *zoom*.

La barra inferior de la aplicación está destinada a proporcionar información sobre el estado de la misma.
Aquí encontramos un texto que indica en todo momento el estado y junto a una barra de progreso que tiene la finalidad de informar en tiempo real de cuántas imágenes se han procesado.

Ante esta pantalla el usuario debe presionar sobre el único botón habilitado, "Importar imagenes". En ese instante aparecerá una ventana emergente pidiendo al usuario que seleccione las imágenes con las cuales quiere trabajar. Este "diálogo" permite navegar por el sistema de archivos para efectuar una selección múltiple de imágenes.
Una vez seleccionadas las imágenes, desaparece el diálogo y se vuelve a la interfaz anterior con la diferencia de que, tal y como podemos observar en la figura Fig. 2, se listan en la zona izquierda todas las imágenes. Dado que el tiempo de carga de las imágenes es prolongado, éstas se cargan en un hilo de ejecución paralelo. De esta manera el usuario puede navegar entre las diferentes imágenes sin esperar a que se hayan cargado las miniaturas de todas.

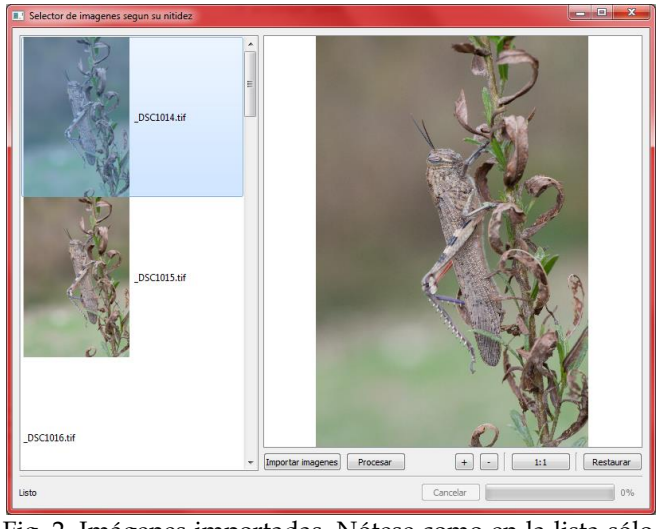

Fig. 2. Imágenes importadas. Nótese como en la lista sólo se ven dos miniaturas. Esto es debido a que el sistema está cargando en segundo plano el resto de imágenes y las reemplaza en cuanto las carga.

Al seleccionar una imagen de la lista, inmediatamente se muestra una versión de baja resolución en el área de visualización y se comienza a cargar la imagen con resolución nativa. Una vez esta carga finaliza, el programa reemplaza la imagen escalada del área de visualización por su versión original manteniendo el nivel de *zoom* y desplazamiento previos.
Una vez cargada una imagen en el área de visualización el usuario puede hacer *zoom* y desplazarse por la misma. El *zoom* puede efectuarse mediante los botones o bien

utilizando la rueda del ratón, mientras que el desplazamiento es bien con las barras laterales o mediante el gesto "arrastar y soltar".

Una vez localizada la zona de interés, aquella en la que se quiere evaluar la nitidez, el usuario ha de marcar un rectánculo utilizando el botón secundario del ratón. Esto se puede apreciar en la Fig. 3.

El último paso por parte del usuario consiste en pulsar el botón "Procesar". En este momento las imágenes se comienzan a procesar en segundo plano, informando al usuario del estado mediante la barra de progreso inferior.
En la barra de estado encontraremos en cada momento el nombre del fichero con mayor nitidez hasta el momento sobre el área seleccionada.
La lista izquierda en todo momento estará ordenada según la nitidez, siendo la primera la más nítida encontrada hasta el momento. Cada vez que el *thread* termine de calcular el gradiente para una imagen se actualizará el orden de la lista. Además la imagen presente en el área de visualización se reemplazará por la mejor imagen encontrada.

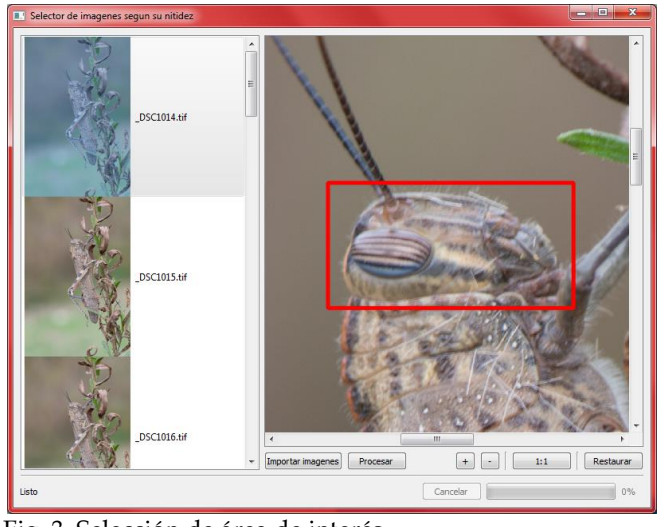

Fig. 3. Selección de área de interés.

6.2 Carga de imágenes

Inicialmente se implementó este módulo del programa únicamente siendo capaz de leer ficheros TIFF de 16 bits por muestra.
Para ello se utilizó la libreria libTIFF. En iteraciones posteriores se añadió la posibilidad de leer TIFF de 8 bits por muestra.
Finalmente se añadió la librería QT como herramienta para lectura de imágenes dada su gran compatibilidad con múltiples formatos de imagen.
La aplicación final por lo tanto es capaz de leer cualquier formato que QT soporte; bmp, gif, jpg, png, pbm, pgm, ppm, xbm, xmp y svg además de tif de 8 y 16 bits por muestra gracias a libTIFF.

Para poder implementar esta funcionalidad sin afectar al resto del programa fue necesario un cambio de representación a la hora de cargar las imágenes mediante QT, puesto que dicha representación difiere de la que utiliza libTIFF. El algoritmo utlizado se encarga de pasar de una representación RGBA a BGRA, tal y como puede observarse en la Fig. 4.

Puesto que hacer este cambio de representación comporta leer y escribir en memoria el tamaño total de cada imagen, se ha procurado implementar de la manera más eficiente posible utilizando operaciones a nivel de bits como son el *bit shifting* o la suma y multiplcación lógicas.

El tiempo empleado para la ejecución de este cambio de representación se encuentra alrededor de 80 milisegundos para una imagen TIFF con 16 bits por muestra y 36 Megapíxeles.

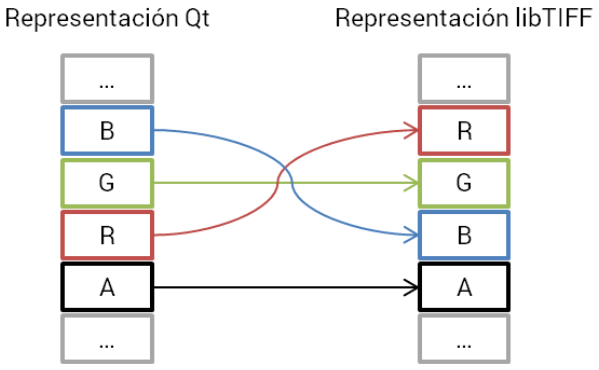

Fig. 4. Cambio de representación interna del color.

6.3 Lógica y procesado

En esta sección comentamos y analizamos los diferentes apartados dentro de la lógica del programa, o del módulo encargado de evaluar la nitidez propiamente.

6.3.1 Nitidez

Para la evaluación de la nitidez, tras haber hecho un repaso por las diferentes métricas de la literatura, optamos por implementar una variación del gradiente. La implementación utilizada consiste en la suma de cuadrados del gradiente.
Se ha elegido esta fórmula puesto que utilizando la potencia se consigue diferenciar más entre imágenes con distintos niveles de enfoque. El algoritmo utilizado para esta evaluación lo podemos ver a continuación.

```
funcion GradienteX(imagen, ancho, altura)

  totalx <- 0

    para y € [0..altura]

      color1 <- ColorImagenEn(imagen, 0, y)
      color2 <- ColorImagenEn(imagen, 1, y)

      totalx <- totalx + (color1-color2)^2

      color1 <- ColorImagenEn(imagen, ancho - 1, y)
      color2 <- ColorImagenEn(imagen, ancho - 2, y)

      totalx <- totalx + (color1-color2)^2

    finpara

    para fila € [0..altura]
      para columna € [1..ancho-1]
        color1  <-  ColorImagenEn(imagen,  columna+1,
fila)
        color2  <-  ColorImagenEn(imagen,  columna-1,
fila)
        totalx <- totalx + ((color1-color2)/2)^2
      finpara
    finpara
```

Como puede observarse en el algoritmo, el gradiente consiste en asignar un valor a cada píxel de la imagen en función de sus vecinos. Para un píxel con posición `x, y`, su valor de gradiente en `x` será la mitad de la diferencia entre el anterior (`x-1, y`) y el siguiente (`x+1, y`).
Esto es así para todos los píxeles de la imagen excepto para los bordes, puesto que carecen de anterior o siguiente. En estos casos el valor del gradiente es la diferencia entre él mismo y su vecino.

Dado que el algoritmo aquí expuesto sólo calcula el gradiente en una coordenada, es necesario utilizar una versión similar para calcular también la métrica en `y`. Una vez tenemos los valores totales de la métrica en `x` y en `y`, sumamos dichos valores para obtener la métrica final.

Aunque inicialmente se planteó el uso de la librería de procesamiento de imágenes OpenCV finalmente se descartó su uso por los pobres resultados en cuanto a la localización de la región de interés.

Por ésta razón la implementación del gradiente es propia y cuenta con la ventaja de consumir muy poca memoria.
Esto es así ya que el resultado se va almacenando en una variable numérica, mientras que si hubiésemos utilizado OpenCV tendríamos que haber calculado el gradiente de la imagen (que ocuparía en memoria como la imagen misma) para después hacer la suma.
Como prueba de concepto se tomó una serie de fotografías a un patrón impreso sobre papel. El objetivo de esta prueba era comprobar el correcto funcionamiento de la métrica. El patrón y algunas imágenes tomadas pueden verse en el Apéndice A.
El experimento consistió en dejar la cámara y el patrón fijos, variando únicamente el enfoque de la cámara. La primera imagen tomada está enfocada lo mejor posible, a continuación se llevó el enfoque a un extremo y después al otro, todo esto haciendo fotos cada poco. Finalmente obtenemos 15 imágenes a las cuales les aplicamos la métrica. En la Fig. 5 podemos ver los valores de la métrica según el número de la imagen en la secuencia.

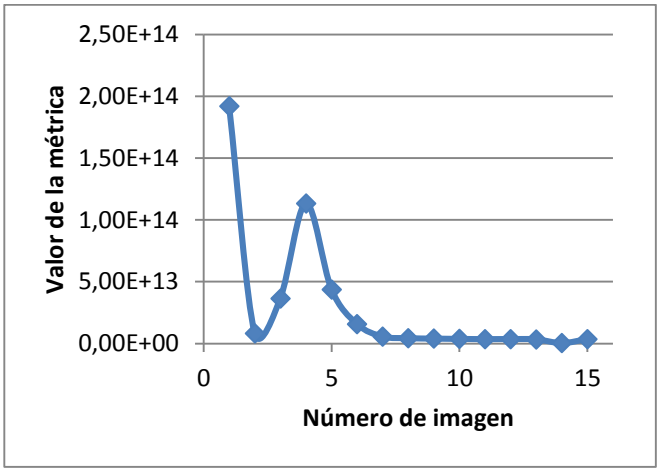

Fig. 5. Valor de la métrica según enfoque

Creemos que la métrica escogida se ajusta muy bien a las necesidades del proyecto. Por este motivo no se ha experimentado con otras posibles maneras de evaluar la nitidez y se han centrado los esfuerzos en otros aspectos del proyecto.

6.3.2 Región de interés

Uno de los mayores problemas encontrados a lo largo del desarrollo del proyecto tiene relación con la región de interés.
A medida que se iban haciendo pruebas e implementando la métrica de evaluación de la nitidez surgió el problema de que al marcar un área por dos puntos en el plano no siempre estábamos evaluando la misma zona de la imagen.
Esto se debe a que a la hora de hacer las instantáneas el fotógrafo mueve involuntariamente la cámara, incluso con la ayuda de un trípode, o bien que aquello que se está fotografiando no está completamente en reposo.
Si analizamos cómo funciona la métrica utilizada podemos ver que el valor de la misma es muy dependiente de la imagen, o zona de la imagen, que se esté evaluando. Por este motivo es muy importante que se evalúe exactamente la misma zona de la imagen, no las mismas coordenadas.
De lo contrario, y para ilustrar el concepto con un ejemplo, podríamos estar evaluando un fondo muy nítido y con un patrón muy marcado (lo cuál hará que la métrica sea muy alta) mientras que el objeto que queremos enfocar y que está en primer plano aparece desenfocado.

Para intentar localizar la región de interés se han utilizado dos mecanismos. La implementación de los mismos se llevó a cabo con la librería OpenCV.

6.3.2.1 SURF

SURF es un tipo de descriptor de características. Su nombre viene de *Speeded Up Robust Features*. Este descriptor de características es conocido por su robustez y especialmente su rapidez de computación en comparación con las características SIFT (*Scale Invariant Feature Transform*).

SURF fue presentado por Herbert Bay, Andreas Ess, Tinne Tuytelaars y Luc Van Gool en 2006 en Austria.

Para llevar a cabo la implementación de la detección de la *Region of Interest* seleccionada en una imagen se utilizó la clase `SurfFeatureDetector` junto con `SurfDescriptorExtractor`. Mediante estas dos clases somos capaces de extraer las características representativas de las dos imágenes entre las cuales hacer el *matching*.

El siguiente paso consiste en descubrir qué descriptores de la primera imagen se corresponden con cuál otro de la segunda. Para esto utilizamos la clase `FlannBasedMatcher` de OpenCV. FLANN se basa en la búsqueda de vecinos cercanos, es decir, se consideran los n parámetros de los descriptores como una posición en un espacio n-dimensional y a partir de aquí se buscan aquellos puntos más cercanos.
Como necesitamos calcular la transformación geométrica entre una imagen y la otra, necesitamos tres puntos de referencia. Tomamos los tres mejores puntos (los tres con menor distancia) y con ellos calculamos la transformación afín usando el método de OpenCV `getAffineTransform`.

Este método nos devuelve una matriz de transformación, la cuál al mutiplicarla por los puntos que definen el área de selección dará los puntos correspondientes en la segunda imagen.
Se ha llevado a cabo una prueba que consiste en la selección de un elemento de interés, en este caso el ojo de un insecto, para después buscarlo en las otras once imágenes del conjunto. Los resultados obtenidos se exponen en la Fig. 6. La imagen con la selección original puede verse en el apéndice B.

Como puede observarse los resultados no son muy satisfactorios, mientras sí que en algunas imágenes se consigue localizar la región de interés, en otras se desvía de tal manera que podríamos evaluar zonas que no nos interesan.

Además cabe destacar que el proceso de localización de la RoI es computacionalmente muy complicado y la ejecución de esta prueba tardó alrededor de 10 minutos.

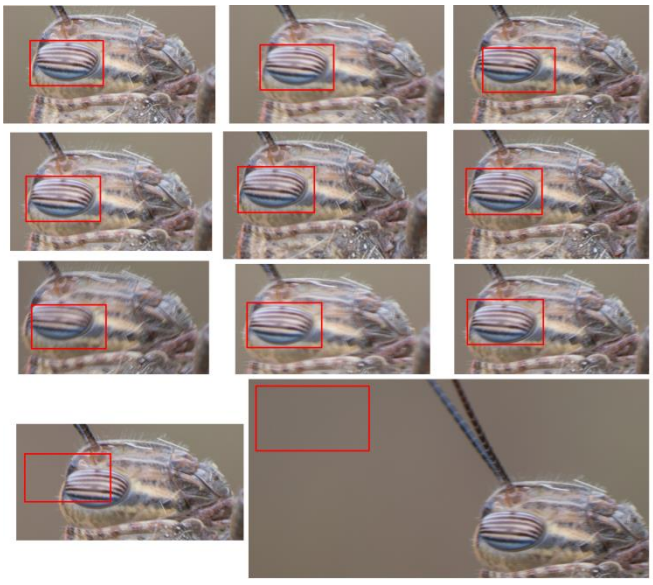

Fig. 6. RoI con SURF

Por estos dos motivos se experimentó con otro método para intentar localizar la región de interés.

6.3.2.2 Transformación rígida

En geometría una transformación rígida es aquella en la cual la pre imagen (antes de la transformación) y la imagen (una vez transformada) tienen la misma forma y tamaño. Existen tres tipos de transformaciones rígidas; rotaciones, desplazamientos y reflexiones.
Puesto que en nuestro caso sólo tenemos desplazamientos y rotaciones (puede haberse rotado la cámara al realizar la foto) en una imagen respecto a la otra, podemos describir la matriz de transformación como una transformación rígida. Esto nos lleva a la segunda técnica implementada para hallar la RoI.

Se ha utilizado la función de OpenCV `estimateRigidTransform`. Dicha función recibe como parámetros las dos imágenes y devuelve la matriz de transformación que utilizaremos para saber en qué punto de la segunda imagen se encuentra uno de la primera.

Esta función internamente detecta que los parámetros son imágenes y extrae las características de las mismas para después utilizar la técnica de mínimos cuadrados.
Esta técnica de análisis numérico se basa en la búsqueda de unos coeficientes para la matriz de transformación con los cuales se minimice el error cuadrático medio (de aquí su nombre).
Igual que con el método anterior utilizamos la matriz para calcular la RoI en las demás imágenes..
Las imágenes utilizadas son las mismas, incluída la de referencia que puede observarse en el apéndice B

Como podemos observar en la Fig. 6 los resultados son algo más consistentes que con la técnica anterior, además de requerir menos código y ser más rapido en tiempo de ejecución.

Aún y así no son lo suficientemente precisos como para considerar éste un buen método.

Fig. 7. Transformación rígida

Como podemos observar en la Fig. 6 los resultados son algo más consistentes que con la técnica anterior, además de requerir menos código y ser más rapido en tiempo de ejecución. Aún y así no son lo suficientemente precisos como para considerar éste un buen método.

El motivo por el cuál no obtenemos buenos resultados radica en la propia diferencia de nitidez. Los métodos probados están pensados para extraer características de una imagen y encontrar esas mismas características en otra.
Al tener imágenes con diferentes niveles de enfoque o nitidez nos encontramos con que las características de las imágenes no son las mismas y por lo tanto no es posible hacer una buena correspondencia.

Visto los resultados de los dos métodos probados nos hemos visto obligados a dejar de lado esta característica, proponerla como posible trabajo futuro y centrar los esfuerzos en otras áreas del proyecto.

6.4 Hilos de ejecución

Otro de los grandes problemas encontrados en el desarrollo del proyecto fue el tiempo de lectura de disco. Trabajando con ficheros de alrededor de 200MB, dependiendo del disco duro del sistema, la lectura puede llegar a demorarse entre 4 y 5 segundos por imagen. Si tenemos en cuenta que trabajamos con unas doce imágenes, lo cuál no es nada descabellado para un uso normal, encontramos que teníamos que esperar alrededor de un minuto para la carga completa de imágenes.
En este caso estamos ante un problema que no se puede solucionar por software, pues la limitación se encuentra en la velocidad de lectura de los discos duros actuales.
Sin embargo tenemos técnicas para intentar mitigar este efecto y poder continuar ofreciendo una experiencia de usuario agradable, pues esperar un minuto no lo es.

Con este problema en mente surgió este módulo de la aplicacion inicialmente no previsto.
Utilizamos *threads* o hilos de ejecución en la aplicación para hacer las dos tareas más pesadas en la misma; la carga y el procesamiento de las imágenes.
Otra limitación que nos encontramos es la memoria, ya que es inviable almacenar en memoria todas las imágenes, por este motivo leeremos las imágenes dos veces al menos.
Al seleccionar las imágenes desde la interfaz, el hilo principal pide al hilo trabajador (*worker thread*) que cargue todas las imágenes en modo miniatura.
De esta manera el hilo de trabajo se encarga de cargar cada imagen, escalarla a una resolución de 1 MP y pasársela al hilo principal para que éste la muestre en la lista y a su vez tenga preparada una previsualización de baja calidad cuando el usuario seleccione dicha imagen.
Al seleccionar en la interfaz una de éstas imágenes instantáneamente veremos en el área de visualización la esta versión de baja resolución. Paralelamente, el hilo principal vuelve a pedir al hilo trabajador que cargue la imagen, pero esta vez sin escalarla. Cuando la carga se completa, la imagen pasa al hilo principal y éste reemplaza la versión de baja resolución por la recibida.
Por último el hilo trabajador se utiliza también a la hora de procesar el gradiente. El *main thread* pide al hilo trabajador que cargue la imagen y calcule el gradiente en el área seleccionada por el usuario.
A cada imagen procesada, el hilo trabajador informa al principal que ha terminado de procesar una imagen y le pasa el valor del gradiente obtenido. Después esta información se utiliza en la interfaz para mostrar en cada momento la mejor imagen procesada.
El thread consta de una cola en la cuál se van añadiendo las tareas pendientes en cuanto le llegan. En una estructura de datos tipo cola normalmente los datos se insertan por el final y salen por el inicio, creando una polítca FIFO (*First In First Out*) pero esto no es siempre así en nuestro caso.
Sólo insertamos por el final en caso de que la petición de trabajo al hilo trabajador sea con la finalidad de cargar una miniatura. En caso de recibir una petición de carga de imagen a tamaño completo o de procesado de imágenes, insertaremos dichas peticiones al inicio de la cola, puesto que consideramos más prioritarias dichas operaciones que no la carga de miniaturas.
Por último, con el fin de que las operaciones de inserción y extracción de la cola sean *thread safe* y no ocasionen problemas derivados de la concurrencia se ha adoptado el uso de un mecanismo de semáforos proporcionado por la librería QT.

Con dicho mecanismo aseguramos que sólo un único hilo de ejecución puede modificar la cola en un momento dado.

7. CONCLUSIONES

Los resultados son satisfactorios. Se ha logrado desarrollar una aplicación que cumple con los objetivos establecidos al principio del proyecto y además se aportan mejoras como la gestión de la carga en segundo plano. Se ha logrado el objetivo de proporcionar una herramienta para acelerar la selección de imágenes semi automatizada.

El resultado de este proyecto ha resultado ser un aplicativo que permite la detección de la mejor imagen y una interfaz para la manipulación de las mismas.

No se ha podido implementar con éxito un método de reconocimiento de RoI suficientemente preciso. La búsqueda de la RoI ha resultado tener mayor complejidad de la esperada y por lo tanto nos hemos visto obligados a obviar dicha característica de la aplicación para así plantearla como posible trabajo futuro.

8. AGRADECIMIENTOS

En primer lugar quiero dar las gracias a mi tutor de Trabajo de final de grado, Francisco Javier Sánchez Pujadas, por su gran dedicación y ayuda, además de proporcionar las imágenes utilizadas para las pruebas.
También cabe destacar la contribución de mi compañero de clase, Samuel Esteban Iglesias, por haber aportado su punto de vista en el diseño de la interfaz de la aplicación.

9. REFERENCIAS

[1] Bay, H. et al., "Speeded-Up Robust Features". Computer Vision and Image Understanding, vol. 110, pp. 346 - 359, 2006
[2] Jaroslav, K, "A new wavelet-based measure of image focus". Pattern Recognition Letters, vol 23, pp. 1785 - 1794, 2002
[3] Subbarao, M., Choi, T., Nikzad, A. "Focusing techniques". SPIE Proceedings, vol. 1823, 1993
[4] "QT Project". Accedido en Febrero, 2014. Disponible: http://qt-project.org/
[5] "TIFF Library and Utilities". Accedido en Febrero, 2014. Disponible: http://www.libtiff.org/
[6] "OpenCV". Accedido en Febrero, 2014. Disponible: http://opencv.org/

10. APÉNDICE

A Patrón utilizado e imágenes tomadas

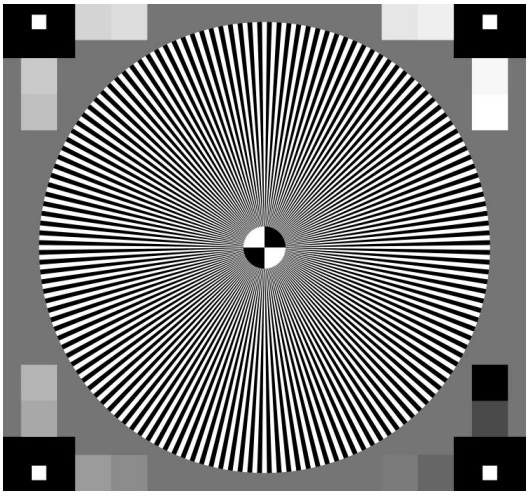

Fig. 8. Patrón

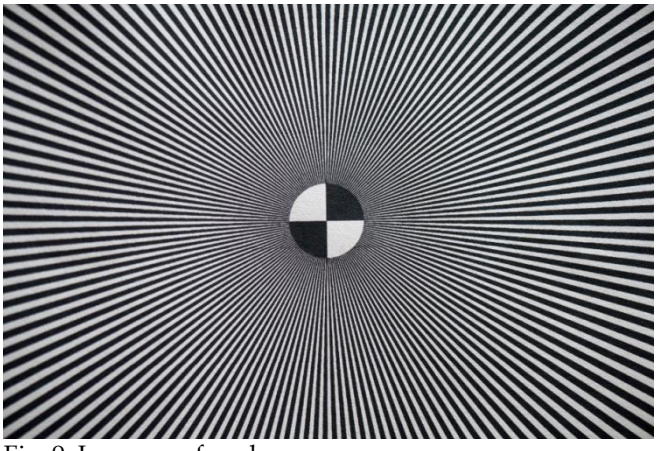

Fig. 9. Imagen enfocada

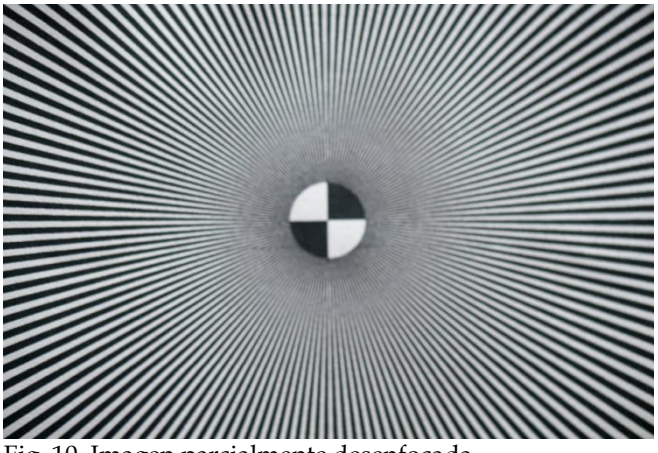

Fig. 10. Imagen parcialmente desenfocada

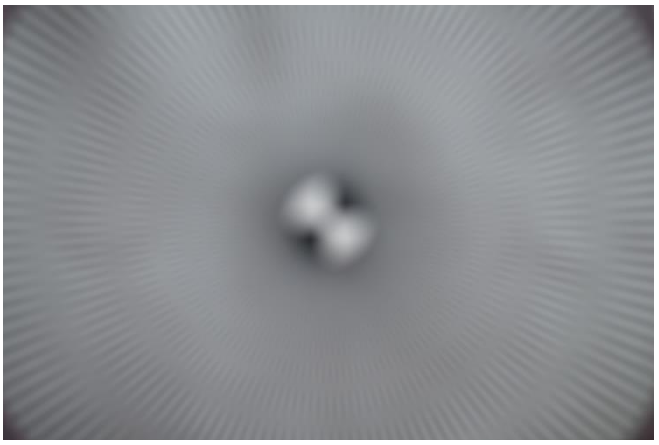

Fig. 11. Imagen muy desenfocada

B Imagen de referencia para las pruebas

Fig. 12. Imagen de referencia con zona de interés marcada